c. specifying a list of property values and event settings

wherein the execution of the objects is determined by [a] the list of property values and event settings.

REMARKS

Before fully responding to the office action, Applicants believe that it would be useful to set forth the definitions of "objects" and "script" used throughout this Response and the claim amendments. These definitions are taken directly from page 9 of the specification.

"OBJECT" shall mean a modular software program written to a defined specification which may perform any function. For purposes of this application, the word "OBJECT" shall be considered synonymous with the term "custom control". Objects are intended to be grouped and assembled into applications.

"SCRIPT" shall mean a set of computer instructions written at a higher level than code ie., farther removed from machine language, which is usually interpreted during execution. "SCRIPT" for purposes of this application shall also refer to the listing of objects and property settings generated by the authoring program of this invention.

Examiner's paragraph 3:

The Examiner objected to the claims noting that "programmatic" was misspelled as "programatic." The amended claims do not contain the word "programmatic."

Examiner's paragraph 5:

The Examiner has rejected claims 1-3, 17, 18, 25, 26, 27, 28, 29, 30, 43, 44, 45, 46, 47, 48, 49, 50, 52, 53, and 54 under 35 U.S.C. § 112, second paragraph as being indefinite for failing to particularly point out and distinctly claim the subject matter. The Examiner noted that

certain terms lack proper antecedent basis. Applicants' response follows each of the Examiners subsection remarks which are underlined:

a. "the objects" in claims 1, 3, 17-19, 23, 26, 29, 44-46, and 54;

Applicants first note that claim 28, not claim 29, was probably intended by the Examiner. Applicants have amended the claims by reciting in the preamble of each claim where appropriate: "employing objects."

b. "the program structure" in claims 2 and 29;

Applicants have amended the claims by reciting in the preamble of each claim: "a program structure."

c. "the four different representations" in claims 2 and 29;

Applicants have amended the claims by reciting in each claim: "a plurality."

d. "the user" in claims 2 and 29;

Applicants have amended the claims by deleting the reference to: "at the election of the user."

e. "those objects" in claim 3;

Applicants have amended the claims by deleting the term "those."

f. "the displayed functionalities" in claims 20, 21, 47, and 48;

Applicants have amended the claims by deleting the term "displayed."

g. "the system" in claims 20 and 21;

Applicants have not amended the claims in response to the Examiners comment. Applicants respectfully submit that the reference to "system" in the preamble "development and run time system" provides sufficient antecedent basis for the reference to "system" found in the subsequent steps.

h. "the choice of objects" in claims 20, 21, 47, and 48;

Applicants have amended the claims to delete "choice of." Reference is now made just to the objects.

i. "the manner of their implementation" in claims 20, 21, 47, and 48;

Applicants have amended the claims to delete the phrase "manner of their implementation."

j. "the values of properties and connecting events" in claims 22 and 49:

Applicants have amended the claims by providing antecedent basis by reciting "having property values and event connections" in the preamble.

k. the settings and connections" in claim 22;

Applicants have amended the claims by providing antecedent basis by reciting "having property values and event connections, which can be set" in the preamble.

l. "the property values and event connections" i claims 22 and 49:

Applicants have amended the claims by providing antecedent basis by reciting "having property values and event connections, which can be set" in the preamble.

m. "the execution of the objects" in claims 23, 27, 50 and 54;

Applicants have not amended claims 23 and 50 in response to the Examiner's objection. Applicants respectfully submit that the reference to "executes objects" in the preamble provides sufficient antecedent basis for the reference to "execution of the objects."

In claims 27 and 54, Applicants have substituted the phrase "dynamically executing" for the term "utilization" in order to provide antecedent basis for "the execution of the objects."

n. "the execution scheduling" in claims 25 and 52;

Applicants have amended claims 25 and 52 to delete the term "scheduling" as redundant since the phrase "managing the execution of objects" encompasses the scheduling of objects.

o. "the temporal relationship" in claims 26 and 53;

Applicants have not amended claims 26 and 53 in response to the Examiner's objection. Applicants respectfully submit that the art of an object oriented parallel processing system inherently requires the management of objects in time. Specifying a "temporal relationship" among the objects is the expression of that parallel processing management. As a well understood term of art, no additional antecedent basis is required.

p. the function of programming constructs" in claims 27 and 54:

Applicants have not amended claims 27 and 54 in response to the Examiner's objection. Applicants respectfully submit that the term "programming constructs" and what is meant by "the function of programming constructs" is a well understood term of art in programming generally, not just in object oriented programming. Further, the term is used in the preamble with its usual meaning.

q. the Examiner has skipped this letter reference in the Office Action.

r. "the application" in claims 27 and 54;

Applicants have removed the reference to "into the application" as having an obvious and redundant meaning.

s. "**the** depiction of the objects" in claims 30;

Applicants have amended the claim to delete the phrase "the depiction of the objects" and have substituted the phrase "the icon for each object in the representations."

t. "the depiction of **the** object" in claims 30;

Please see response to subsection s.

u. "the work of programming constructs" in claims 43 and 46;

Applicants believe that the Examiner intended to refer to claims 42 and 16, both of which contain the phrase "the work of programming constructs". Claim 46 does not contain the phrase. Applicants have amended claims 16 and 43 to recite "the function of programming constructs" to make the claims consistent with the other claims reciting "the function." Applicants have not further amended the claims and refer the Examiner to the reasons stated under the response to subsection "p" above.

v. "the run time execution of the objects" in claim 52;

Applicants have deleted the term "run time" as being obvious and redundant.

Examiner's paragraph 7:

The Examiner has rejected claims 1-54 under 35 U.S.C. § 102(e) as being anticipated by Smith. Before responding to each of the Examiner's references, Applicants believe that some general background with respect to the Smith patent, object oriented programming, and Applicants' implementation of their invention in an object oriented programming environment will aid the Examiner's understanding of Applicants' responses to the Examiner's remarks. In particular, it is important to understand that the standardized "objects" (objects written to a defined specification and published for third party use), which were originally designed to be used in a compiled programming system such as Microsoft's Visual Basic, are used by Applicants in an interpreted system which does not require compiling.

Patent 5,485,618 "Methods and interface for building command expressions in a computer system" is a patent on a syntax generator. This invention builds single command lines

according to rules encoded into the system. Once these command expressions are built according to the defined syntax they are placed into a standard programming language. An interesting aspect about the system is the tokenization of operands and operators that allows incomplete expressions to be built while the rules are maintained.

The "Expression builder" is either added in as part of an existing programming system, like dBASE, C++, or Visual Basic, or it operates as a parallel application, but in either case it is a line editor that supports the creation of standard text based development systems. It puts place holder tokens into expressions that are being constructed by the user as the user builds the expression. These tokens are not software objects of the kind referred to in the present invention. They are text place holders used to maintain correct syntax in one part of a command line, while the user edits another place in that line.

In contrast the present invention is a complete stand-alone program used to develop other end user applications that can run locally, over a network, or over the Internet. It uses four compete user interfaces to aid in the construction of those end user applications. Each user interface is fully capable as a development interface on its own, however greater utility is gained by allowing the developer to select the interface that is most suitable for the project. The developer can view the project in one of the other interfaces at any time to get a different view of the project. He can even run the application in preview mode and follow the execution in any or all of the four development views at the same time.

The present invention is a kind of program called a Container that uses software objects as building blocks for the creation of end user applications. The present invention uses objects that are like self-contained subprograms. These objects take parameters as inputs. These parameters are called the "properties". The property values of an object may change as the

result of an object executing. The present invention allows users to store special @functions within properties. The present invention interprets those @functions to move data between the objects according to the design of the developer. For example, an edit box object can @get a text value from a database object with the help of the present invention container. Objects also issue signals (called events). These signals (events) tell the container that the status within the object has changed. Such an event signal tells the present invention to look within the object and see which other object to activate. In this sense, the present invention acts like a traffic-cop. Each object has its own list of events that it generates. These events may include something like "finished" to show that the processing task has been complete, or another event may be "error" to show that the task has not completed property. It may be useful to think of properties as input and output values that come from and go to the container, and events as signals to the container which tell the container about the status of the object. The unique nature of the present invention required the present invention container to wrap the standard objects with new properties and events so that it can manage the objects. For example the container of the present invention needs to know when an instance of each and every object has been successfully created. That is why in the present invention there is a "Branch on realize" event added to each object. The container of the present invention needs an internal naming scheme. Therefore, a "_Label" property was wrapped around each object. This allows the container to recognize each object by a unique identifier placed into this property value.

When an application developed within the present invention system is deployed, it is delivered as a script that is executed within the run time environment of the present invention. The run time environment is a container for objects that allows those objects to communicate with one another, and to instantiate one another according to events that occur within the objects.

A unique aspect of the present invention is that, unlike standard programming systems which execute a program that in turn tells the objects what to do, the present invention puts the relevant commands inside the objects so in effect the objects tell each other what to do. Another unique aspect of the present invention is that it provides a container for the objects which interprets the script to control the objects according to the values stored in the script. The prior art applications that used such objects were compiled and delivered as executable programs. Additionally because in the present invention system there is no program that commands the objects, it is not necessary for the present invention to have program constructs embedded within the container. It becomes logical that, since the objects are commanding one another, there should also be objects that act as program constructs. These construct objects can be added or subtracted as needed by the developer.

Applicants will now respond to the specific bases for rejection made by the Examiner.

Examiner's paragraph 7, page 5:

Per claims 1 and 28, Smith discloses at least:

means for wrapping standardized objects with additional properties and events beyond those properties and events provided in the standardized object (see at least column 3, lines 32-37; column 4, lines 8-19 and related discussion elsewhere in the specification; and

Applicants submit that the claims are not anticipated by the disclosures in column 3, lines 32-37, or Col 4 lines 8-19. Col 3 lines 32-37 pertain to the elements of the user interface to the "Expression Builder" including such standard elements as an "Undo" button. These user interface elements are not new properties that wrap existing standard objects, nor are they events that wrap the same software objects. Col. 4, lines 8-19 pertain to the reaction of the "Expression Builder" to user actions. These lines describe what the "Expression Builder" does when the user

cuts, pastes, deletes, tokens while be builds his expression. The "Expression Builder" inserts and deletes tokens and other elements to the command string in reaction to the entries of the user. This is done to maintain the correct syntax, even if that syntax is maintained with text placeholders called tokens. Col. 4, lines 21-61 merely set forth the rules used by the Expression Builder to determine if the syntax in the presented line is correct or not. Applicants respectfully submit that none of the referenced citations has anything to do with using standardized objects or wrapping such objects with additional properties or events.

means for utilizing the additional properties and events to link and sequence the objects into the application (see at least column 4, lines 21-61 and related discussion elsewhere in the specification).

Smith's invention is a line editor that insures the correct syntax of a single line of computer code for use in a second system. Smith's invention does not necessarily deal with software objects, and when it does, only exposes the properties and events available in the standard object to the developer so the developer can write a single line of code. No properties beyond those available in the standard object are available to Smith so it is impossible for Smith to use "...additional properties and events to link and sequence the objects in the application" as the Examiner has suggested. Smith does not link or sequence objects at anytime. That is done by the application with which Smith's invention works. Smith does not add new properties and events to objects either in the process of development, nor in the code that is developed through the use of this invention.

Examiner's paragraph 7, page 5:

Per claims 2 and 29, Smith discloses at least:

means for simultaneously displaying different representations of the program structure (see at least Figure 4E and related discussion in the specification); and

Figure 4E shows the same user interface window at different times in different states. This Figure represents the result of editing a "0" and making it a "2" and also changing a token "expN" into an actual value "3." This does not represent the program structure, rather it shows the process of editing a single command line within a larger program structure that can not be determined by the Figure.

means for manipulating the program structure within each of the four different representations; wherein the representations of the program structure may be synchronized among displays at the election of the user (see at least Figure 2C and related discussion in the specification).

Figure 2C is a full screen view of the "Expression Builder". It is clear that there are several windows opened at once, however none of those windows represents the structure of the program. They are all dedicated to the construction of a single line of program code. The kind of code to be generated is selected from the leftmost window from which a user selects the object to be commanded by the code. The lower right window is the "Expressions Builder" dialogue box that is shown when the user selects the "Build Expression" from the Edit menu. The current result of the process of building the single line of computer code called an expression is echoed into the top right window called "Command." Since all these windows focus on the construction of a single line of computer source code, there is no view of the overall structure of the program, let alone multiple views of the overall structure as the present invention provides. Applicants respectfully submit that the cited references do not anticipate claims 2 and 29.

Examiner's paragraph 7, page 6:

Per claims 3 and 30, Smith further discloses

a means for highlighting the depiction of the objects in the representations as those objects are being realized during application development playback preview (see at least Figure 4C, block 432 and related discussion in the specification).

Figure 4C, block 432 is an edit box that allows the user to enter values into the expression while that expression is being built. This edit box must be closed before the expression is complete. The expression must be complete before program execution can proceed. Therefore, it is not a highlight of any element of the program that is visible during program execution. Furthermore, no objects are depicted. Accordingly, Applicants respectfully submit that the cited reference does not anticipate claims 3 and 20.

Examiner's paragraph 7, page 6:

Per claims 4, 8, 12, 31, 35, and 39, Smith discloses at least:

a development environment and a playback environment that have no logical operators (see at least Figure 2B and related discussion in the specification); and

means for utilizing, by specifying property values, standard objects (see at least 2A and related discussion in the specification).

Figures 2A and 2B do show a development environment for creating a series of program expressions. However, by virtue of the very type of program that the Expression Builder is, included in those expressions are logical operators. Refer to the window labeled Command in Figure 2B. The "=" in the expression is an assignment operation, and in Figure 3B the right list box labeled "Paste" has a list of logical operators. These operators are placed into the

expression that the Smith invention generates. Those operators are later used by the development environment in the further construction of the expressions and also in the playback environment as part of machine-readable code.

Most importantly, the present invention does not have these operators as part of the container, rather it uses software objects which contain these operators. In this way the present invention can be added to and modified as conditions warrant as easily as the addition of any object in the system (a 2-minute process conducted by end users). Changes to program operators in the system shown in Smith's invention would entail a rewrite of the invention and the compiler that turned the expressions into machine code by the vendor.

Figure 2A in the "Expression Builder" might be used for specifying property values, although the Figure does not demonstrate this happening. Instead Figure 2A shows the beginning of the process of building an expression to command the performance of a task. This is fundamentally different than the way the present invention works. Instead of making a long list of commands that are compiled into a program and later executed in sequence, the present invention moves all command operations into the objects themselves. The objects command themselves and each other. The present invention acts as an intermediary or traffic cop that realizes (makes an instance) objects and lets them run according to their own nature. At the start the present invention realizes the first object and once it is running that object tells the present invention to start other objects running based on the results of the its work. Each object in turn starts or stops other objects. The present invention also connects the objects as they send data between one another. The difference between the Smith invention and the present invention is the difference between a compiled system, and an interpreted system. The Smith invention is for use in creating code for compiled systems - C, Pascal, and C++ are mentioned as examples

(Col. 1 lines 43-44, 53; Col. 2 line 36; Col. 11, lines 60-63; Col. 15, lines 41-67) while the present invention is by its very nature a system that requires the code that sits within the objects to be interpreted as program execution continues.

The specification of property values is required in object oriented software programming systems. Applicants do not claim to have invented the art of object oriented programming, rather Applicants claim to have invented the means of using these objects within a uniquely designed container by setting the property values from within the objects themselves rather than from a series of commands compiled into an application that is later linked and stored as an executable file. Accordingly, Applicants respectfully submit that the cited reference does not anticipate claims 4, 8, 12, 31, 35, and 39.

Examiner's paragraph 7, page 6:

Per claims 5, 9, 13, 32, 36, and 40, Smith further discloses

a means for communicating among standard objects through the exchange of property values (see at least Figure 4D and related discussion in the specification).

Figure 4D shows a step in the process of creating an expression. As stated above, such expressions are designed for use in larger programs written for later compilation. Neither this Figure nor the accompanying discussion demonstrates how one value moves from one object to another in an interpreted environment, as the present invention does. It only shows an example of how a syntactically correct statement is created according to rules that apply to the rules of compiled languages. Smith's invention solves common problems inherent to compiled software systems, but these problems do not exist in the present invention. In this regard, Smith teaches directly away from the present invention.

For example, one of the problems solved by the Smith invention is the problem of data typing. (Col. 2, line 28 - Col. 3, line 5). There is no data typing in the present invention container. Data has to be typed by the objects as the objects use the values sent to them. Another problem solved by Smith is the "plethora of 'functions' and 'reserved words'" (Col. 3, line 5-7). By contrast, there are fewer than 10 functions in the present invention. The most used are @Get and @Set. The @Get function fetches a property value from one object and puts it into a property field in the @Getting object. The @Set function changes the value of a property in a second object from the first. The work done by the objects is accomplished by setting the properties of those objects at design time or @Setting them at run time. There is no indication that Smith's invention anticipates this kind of interpreted transmission of values between objects. Accordingly, Applicants respectfully submit that the cited reference does not anticipate claims 5, 9, 13, 32, 36, and 40.

Examiner's paragraph 7, page 6:

Per claims 6, 10, 14, 33, 37, and 41, Smith further discloses

a means for communicating among standard objects wherein an event generated by a object triggers an instance of another object (see at least Figure 4E and related discussion in the specification).

Figure 4E shows how the "Expression Builder" changes one part of a string based on user initiated changes in another part of the string. The Expression builder makes text strings according to set syntactical rules, and does not directly address the manipulation of objects. The claim language above addresses the present invention's ability to create an instance of one object based on an event that occurs in a second object. This is very different from inserting text tokens

into a string. The objects with which the present invention deals are full featured sub-programs, not individual words in a larger text string such as the tokens in Smith's Expression Builder Figure 4E. Accordingly, Applicants respectfully submit that the cited reference does not anticipate claims 6, 10, 14, 33, 37, and 41.

Examiner's paragraph 7, page 6:

Per claims 7, 11, 15, 34, 38, and 42, Smith further discloses

a means for communicating among standard objects wherein an event generated by a object triggers an instance of another object (see at least Figure 4E and related discussion in the specification).

Figure 4E shows how the "Expression Builder" changes one part of a string based on user initiated changes in another part of the string. The Expression Builder makes text strings according to set syntactical rules, and does not directly address the manipulation of objects. The claim language above addresses the present invention's ability to create an instance of one object based on an event that occurs in a second object. This is very different from inserting text tokens into a string. The objects with which the present invention deals are full featured sub-programs, not individual words in a larger text string such as the tokens in Smith's Expression Builder Figure 4E. Accordingly, Applicants respectfully submit that the cited reference does not anticipate claims 7, 11, 15, 34, 38, and 42.

Examiner's paragraph 7, page 7:

Per claims 16 and 43, Smith further discloses

a means for adding additional programming constructs by employing standard objects that perform the work of programming constructs wherein unlimited expansion of the program

capabilities is achieved (see at least Figure 4A and related discussion in the specification).

Figure 4A does illustrate the list-based nature of Smith's invention and the discussion covers how the system uses the rules of a known computer language as basis for the list elements. Smith teaches that the list structure is determined by the syntax rules of the language since the easy creation of language expressions is the reason for the invention. Smith does not teach extending the languages in question with new constructs, or even new functions. Col. 13, lines 6-18 describes how built-in (system) variables, and user defined local variable names can be included in these lists to facilitate the construction of the expressions that use existing functions, and programming constructs. Smith does not teach extending the system with new programming constructs in his invention. Smith's invention makes the construction of expressions easier, but those expressions must conform to an existing language syntax. In the present invention, Applicants have taught how to add constructs by employing objects that perform the work of constructs. Accordingly, Applicants respectfully submit that the cited reference does not anticipate claims 16 and 43.

Examiner's paragraph 7, page 7:

Per claims 17, 18, 19, 44, 45, and 46, Smith discloses at least

a run time program that has no logical operators (see at least Figure 2B and related discussion in the specification); and

Smith's invention has no run time system, it is an "Expression Generator" than helps create expressions in the correct syntax. Smith's system, itself, does not "run" objects, rather it creates source code used in other applications. These other applications may "run" the objects, but Smith does not teach running objects. Also, Figure 2B shows the existence of a logical

operator "=" in the window labeled "Command". In Col. 7, lines 35-36 Smith teaches that the syntax shown in this system is defined, "...as mandated by the target language." The particular language cited by Smith is the Dbase language. According to a standard reference in the field, CMP Publishing, the definition of a programming language is:

A language used to write instructions for the computer. It lets the programmer express data processing in a symbolic manner without regard to machine-specific details.

The statements that are written by the programmer are called "source language," and they are translated into the computer's "machine language" by programs called "assemblers," "compilers" and "interpreters." For example, when a programmer writes "MULTIPLY HOURS TIMES RATE", "MULTIPLY" must be turned into a code that means multiply, and "HOURS" and "RATE" must be turned into memory locations where those items of data are actually located. Other standard operators include arithmetic ones (+, -, *, /) and logical ones (if, not, else) etc. Since computer systems turn human readable code into machine-readable commands, all programming languages included standard operators; that is, until the present invention was developed.

A second definition of a computer language from the Department of Computing Imperial College of Science, Technology and Medicine, University of London defines:

> Programming language: A formal language in which computer programs are written. The definition of a particular language consists of both **syntax** (how the various symbols of the language may be combined) and **semantics** (the meaning of the language constructs).

The constructs mentioned in the above definition include logical operators as stated before. Smith's invention is useful only with a target programming language of known scope and syntax

as defined in both definitions above. At the time of Smith's invention all the examples of target languages he gives, C, C++, Pascal, Dbase etc, had logical operators as part of the executable code. Even known interpreted languages like Basic had logical operators imbedded into the interpreter so the commands could be read and turned into machine executable commands.

Smith does not teach the Building of Expressions without constructs like logical operators because such expressions would be meaningless to the target languages his system was designed for. Instead Smith's system makes correct programming easier to accomplish by insuring the correct syntax and selection of constructs according to the known definition of the target computer programming language. Even if one were to argue that Smith's invention includes languages without language constructs, it still does not anticipate the present invention where any language construct can be added or subtracted at will. Smith's design of and reliance upon a rule based system makes it impossible for Smith's invention to encompass or adapt to the present invention since such rules are impossible to maintain when constructs can be added and subtracted at will by the user as is possible with the present invention. As stated earlier, the ability to add local variables to a drop down list does not imply the addition of constructs because variables are just names for values while constructs have their own rules. Smith does not show how such rule-based constructs may be added because of the complex nature of establishing the rules. Smith's patent describes the complexity of creating such rules and does not teach that these rules and the constructs they describe can be easily or casually added.

means for utilizing standard objects by identifying the objects and specifying property values (see at least Figure 2C and related discussion in the specification);

Figure 2C shows the user interface for building expressions based on a syntactically defined language. This is not equivalent to using standard objects for language constructs and

specifying the property values required putting those objects to use. There is nothing in Smith's invention that anticipates the use of standard objects as language constructs. There is no indication that Smith's rules-based system would be robust enough to provide the flexibility to incorporate the use of standard objects to implement additional language constructs without the underlying computer language compiler.

It is not enough to simply state "The system and methods of the present invention may be advantageously applied to a variety of system and application software, including electronic spreadsheet systems, word processors, and the like." (Col 6, lines 14-16) to make it apply to a system that is radically different than "the like" and was not explicitly anticipated by Smith. Smith's work clearly teaches the art of a syntax generator with tokens, not the kind of new system the present invention represents. Figure 3B shows the creation of a program expression to be converted to compiled or interpreted machine code as part of a larger syntactically based source code. The reason the " < " symbol is listed is because it is part of the rule set of the language for which the expression is to be generated. In software of the present invention the " < " construct is not part of some larger rule based syntax of the target programming language, rather it resides only in a construct object. Such a construct would remain within the construct object and not be compiled at all. For example in the present invention, if the construct object were removed, there would be no " < " construct. In a compiled language, such as anticipated in Smith, the program compiler would have the " < " construct as part of the base code, even if it were never called. The same could be said for an interpreted language like Basic, or even an Excel macro. The Interpreter contains the ability to understand all the constructs possible within the parameters of the target language syntax even if the construct is never used. The same is not true for the present invention where these constructs are added and

subtracted at will and are encapsulated within standard objects.

Examiner's paragraph 7, page 7:

Per claims 20, 21, 47, and 48, Smith discloses at least:

means for instantiating objects (see at least Figure 4A and related discussion in the specification);

Figure 4A does not represent the instantiation of objects. Figure 4A represents the process by which one uses Smith's "Expression Builder" to write a line of code. Smith does not create an instance of any object in the process. Instantiation is a word of art in object oriented programming. Smith's invention is not an object oriented program. It may be possible for a developer to use a Smith-like system to write code that results in a program which creates an instance of an object, but Smith's invention does not, itself, have the capability of instantiating an object.

means for integrating objects (see at least Figure 2C and related discussion in the specification);

Figure 2C does show a means for integrating objects. Applicants do not claim to have invented integrating objects into software. What applicants do claim is a new, unique, and non-obvious method of utilizing objects. Smith teaches a means of creating a line of program code according to the rules of a known target language. That language may be used to integrate objects into a software application, but that does not mean that Smith teaches how to integrate those objects. Smith does not actually integrate objects beyond the capabilities of the base application which Smith's invention "helps". Smith's program relies on an underlying base program to provide any object integration. In the prior art, objects could only be integrated when

the base program written in the target language is compiled. Smith is supplying a line editor to make it easier to write the program code for the underlying system to do this integration. By contrast, the present invention is an interpreted system that does integrate objects both during the development stage and during run time execution.

means for sequencing objects (see at least Figure 2B and related discussion in the specification);

Figure 2B does not show a means for sequencing objects. In fact, all Figure 2B and related text teach is a typical sequence of operations the user of Smith's program undertakes when generating a syntactically correct expression. No "objects' as the meaning is understood in object oriented programming are involved. The Smith generated expression may result in a program which places an object in a certain sequence in the resulting program, but that is not the same as Smith's program making such placement. Figure 2B does not represent the sequencing of objects in a container without logical operators, since such a container would be useless to a Smith type system as mentioned above. Smith's system is a Helper Application that runs along side the development system and acts as a line editor.

means for providing communications among objects (see at least Figure 2C and related discussion in the specification); and

Figure 2C does not show communication among objects, rather it shows how Smith's invention generates expressions using tokens as placeholders. At the risk of being repetitious, Applicants again point out that the Smith patent does not teach the use of objects as that term is understood in the object oriented programming art.

Further, the specification at Col. 8, lines 15-34 clearly teaches that the Expression Builder is not a stand alone program but rather a "helper application" that is used to create single

expressions upon demand. The preferred embodiment (Col. 8, line 16) is demonstrated for creating single expressions, not complete programs. The example "Command" window found in Figure 2C is not even part of Smith's invention. The "Command" window is part of the dBase system. Smith's invention interacts with the Dbase system, but is separate from it. Smith's invention is a line editor with rules for a particular target language (Col. 8, lines 14-15).

means of synchronizing views wherein the displayed functionalities performed by the system during execution are determined by the choice of objects use(d) and the manner of their implementation in the system (see at least Figure 4C and related discussion in the specification).

Figure 4C shows the use of an edit box into which a developer types information. That information is placed into the token "". This token is a placeholder for the text data entered into the edit box Ref. No. 432 in Figure 4C. This is discussed at Col. 11, lines 32-38. Since Smith's invention is a line editor, it does not contain even one view of a complete project, only one expression within that project. Since not one complete view is visible, it is impossible to say that multiple views are synchronized. Even if, for the sake of argument, one assumes that Smith had multiple views synchronized, which he does not, Smith does not display the functionality of any work performed by the expression. If any functionality is displayed, that display is done by the program which Smith's invention is helping. In the case discussed in the patent, the program that would do this display is dBASE. dBASE, however, does not display the functionality either. Applicants' prior comments with respect to object oriented programming objects are repeated here by incorporation.

Examiner's paragraph 7, page 8:

Per claims 22, and 49, Smith discloses at least:

means for setting the values of properties and connecting events (see at least Figure 2C and related discussion in the specification);

Figure 2C shows how the Smith line editor is started from the main (or parent) program. Smith's invention may be integrated into dBASE, or run parallel to it and it is made available with a menu choice. As mentioned before dBASE, C, C++, Visual Basic and other application building tools had the ability to incorporate standard objects as part of the complete application. However they did not use the same means for setting values of properties nor did they use the same means for connecting events found in the present invention. All of the prior art systems were compiled based on a known set of language constructs or reduced to an intermediate machine code variant such as "p" code. Applicants refer the Examiner to Applicants' earlier comments on languages and constructs; how the present invention moves "@" functions into the properties of the objects; and how the present invention monitors events and uses information associated with the object to connect objects with one another according to event type.

means for recording and maintaining a history of properties settings and event connections as the settings and connections are changed (see at least Figure 2A and related discussion in the specification); *and*

means for traversing the history one change at a time wherein the property values and event connections may be edited from any point in the history (see at least Figure 2A and related discussion in the specification).

The undo feature as implemented in the present invention was unique to the present invention at the time of its invention. The present invention supplies end users with the ability to step back through their work with a step by step granularity up to 300 steps. The end user developer can choose to set the undo level from 1 to 300 steps as he/she sees fit.

Figures 2A - 2C. There is an undo feature mentioned at Col, 3, line 33 and Col. 9, lines 40-43. There is no indication of a granularity to this feature, nor is there any way to set the number of undo steps mentioned. As indicated at col. 9, line 40, it appears to be a single step undo as is indicated: "If a mistake is made while creating an expression, the user can click "Undo" button 303 to undo the last operation; the button is dimmed (inoperable) if no previous "undoable" editing action exists."

It is well known by those in the programming art, that it is often difficult to catch errors right after they are made. Consequently, a one step undo feature is of limited use. The present invention's 300 levels of undo function provides an expanded ability not previously described in the prior art and can not be made obvious by a one level undo feature. The ability to go back through time to revisit a significant number of earlier stages in program development on a step by step basis is a significant advance in the art of object oriented programming tools because errors frequently don't make themselves obvious until many steps are taken. The ability to go back and find the mistake speeds development measurably. No where is this extensive ability taught in the prior art of object oriented programming.

Examiner's paragraph 7, page 8:

Per Claims 23 and 50 discloses at least:

means for wrapping standardized objects with additional properties and events beyond those properties and events provided in the standardized object (see at least column 3, lines 32-37; column 4, lines 8-19 and related discussion elsewhere in the specification);

Col. 3, lines 32-37 are a mere description of the elements of a user interface do not make the wrapping of software objects with new properties and events obvious. Further, Col. 4, lines

8-19, the use of tokens in a line editor that adjust to changes made by the user do not make wrapping of software objects with new properties and events obvious. Wrapping of software objects in object oriented programming is an entirely different process. Applicants' again respectfully incorporate their prior responses with respect to the nature of software objects in the object oriented programming art, specifically the background comments provided at the beginning of this Response with respect to wrapping objects with new properties and events, and the difference between text place holders (called tokens) and programmable software objects.

means for utilizing the additional properties and events to link and sequence the objects (see at least column 4, line 21-61 and related discussion elsewhere in the specification);

Col. 4, lines 21-61 provides a detailed description the "Expression Builder" uses in the creation of lines. It delves into the rules behind the insertion of tokens, the use of "cracks" and the way such elements are controlled in the display of expressions before the expression is completed to insure that the complete expression is syntactically correct. This explanation of a useful line editor has nothing to do with the way the program of the present invention treats programmable software objects. In every case covered in the "Expression Builder", the syntax that is created adheres to the rules of the programming system (target language) that it helps. The syntax options that are displayed based on the on-going efforts of the developer are all predetermined by the syntax of the programming system into which the expression is to be inserted upon completion. These tokens, "cracks", and other symbols are neither equivalent in any way to the addition of the new properties with which the present invention wraps standard programmable software objects, nor are they analogous to the new events with which the present invention wraps standard programmable software objects. The pieces of text that are added and subtracted in the edit box in 4C [431] are text place holders for syntax elements that are placed

into the expression on an interactive basis to insure that the developer does not forget to complete the expression correctly. This is in contrast to encapsulating a software object with a wrapper, in effect sub-classing that object, within a software container in such a manner that the new properties and events interact with the container as if they were part of the original component.

means for reading one or more sets of property values maintained separately from the run time system and the objects wherein the execution of the objects is governed by the property values (see at least Figure 2B and related discussion in the specification);

The current invention allows a developer to create a script, and store that script for recall at a later time. The script contains the settings of all the property values and event tables for the objects as the developer determined he wanted to be implemented upon subsequent execution. When an application developed with the system of the present invention is to be run, the script is recalled and loaded into the container. When execution starts, the values recorded in the script are loaded into the objects. The objects use that information to perform their tasks within the container.

Figure 2B shows the result of assigning a value to a variable within the context of the helper application invented by Smith. This does not teach the means for reading a script that contains sets of property values and interpreting that script in such a way that the values in that script are used to control the operation of objects. The current invention was, at the time of its invention, unique in its ability to use objects (written to a defined specification) in an interpreted environment. Other programs that used these objects were compiled. Only the present invention was able to read scripts interactively according to direction from users or the logic of the initial program and create instances of objects on demand from this interpreted script based on the

script alone or in combination with user interaction.

Examiner's paragraph 7, pate 9:

Per claims 24, and 51, Smith further discloses

means for adding programming constructs and sub-languages utilizing objects (see at least Figure 4A and related discussion in the specification).

Figure 4A does not accommodate the addition of programming constructs utilizing objects, rather it teaches the opposite. As stated above, the very nature of the rule based system found in Smith's "Expression Builder" would make the addition of programming constructs virtually impossible. Smith teaches no method for adding to the rule system, and this rule system is the basis for the "Expression Builder" and such additions would be required when such constructs were added to Smith's system. On the other hand, the software of the present invention accommodates such additional constructs by allowing these constructs to be added to the system as objects.

Examiner's paragraph 7, pate 9:

Per claims 25, 52, and 54, Smith discloses at least:

means for wrapping standardized objects with additional properties and events beyond those properties and events provided in the standardized objects (see at least column 3, line 32-37; column 4, lines 8-19 and related discussion elsewhere in the specification);

Col 3 lines 32-37 is an example of the use of buttons, list boxes and other elements commonly found in dialogue boxes. While it is clear that the edit box of Smith's invention is surrounded by these visual elements, there is no similarity between that dialogue box and the art of wrapping common objects with new properties and events as taught in the present

invention. The present invention teaches a method of encapsulation (subclassing) of software objects which were written to a defined specification and distributed to third parties for their use. The comparison suggested by the Examiner is not valid. The known art of constructing common dialogues does not teach subclassing of objects.

Col 4 lines 8-19 The invention of a method of subclassing objects is not made obvious by the use of placeholders in a line editor. Prior to the development of the present invention, many objects, such VBX controls, were written to a known and defined standard and made available for use by third parties. These objects were considered to be less useful than objects custom written for use in individual programs because the latter type of objects could be subclassed at the time of their creation. It was generally believed at the time that objects like VBX controls could not be subclassed and subsequently they were considered less useful in the art of programming. The present invention teaches a method to subclass these objects and use them in a novel manner thereby greatly extending the value and utility of the objects.

means for utilizing the additional properties and events to link and sequence the objects (see at least column 4, line 21-61 and related discussion elsewhere in the specification);

Col. 4, lines 21- 61. Rules used in the dynamic adjustment of placeholders in a line editor bears no relation to the present invention's use of additional properties and events of subclassed standard objects to link and sequence objects. Such line editor rules can not render the present invention obvious.

means for specifying property values (see at least Figure 2C and related discussion in the specification); *and*

The means used to specify property values in the present invention is vastly different from the means taught by Smith. As noted above, Smith's invention does not employ objects.

The means for specifying those values in Smith's invention is the way the Smith's invention provides for the specification of property values. By contrast, the present invention specifies the property values through a property browser using special "@" functions known as "@Get" and "@Set." These functions are stored and used within the objects themselves. This method is very different than the standard code based systems that Smith's invention assists. Smith's invention provides for specification of a property value only when the target program language requires such a value.

means for saving the property values to a separate file wherein the run time execution of the object is determined by the property values (see at least Figure 2B and related discussion in the specification).

Figures 2B and 2C show a means for specifying property values that is very different than the means used in the present invention. The means used in Smith's invention is through a line editor that relies on a (third party) programming system. Smith's invention does not teach supplying property values any more than it teaches the creation of any other standard expression in a programming language. The supplying of property values is well established within the known universe of standard programming systems. An object oriented programming system that does not provide for the specification of property values would be of little use.

What is most important is the way those property values are specified and used are unique in OZONE the present invention. Such use is not anticipated by the prior art.

In the development environment of the present invention, the user interface relies on a object browser that lists the property values and provides an event table for matching an event with the target object that is to be instantiated when the event occurs (such sue of an object browser is not unique to the present invention). Most importantly, however, the present

invention allows developers to insert @functions into the property values listed in the object browser. These @functions instruct the container how to make data flow between the objects at run time. The present invention script is a record of the property values as they were set by the developer at design time, and in the case were an @function was inserted into a property value, that @function is stored in the script for later execution. The difference between a standard programming system with expressions that are developed according to a complex syntax which requires a Smith type helper tool, and the present invention could not be greater. The present invention puts functions and branching information in the record of the object. When the object is run, "instantiated", those values are loaded from the script into the object. When the loading process encounters an @function the present invention container moves the data between the various objects as instructed. This effectively moves all the control code into the objects themselves. On the other hand, Smith's "Expression Builder" makes statements for standard systems like C++, dBASE, etc. Standard systems like C++, dBASE, etc. take the opposite approach and have the program that has been compiled from the expressions run objects according to rules set out in the code. The present invention does not have a step by step text based program which describes the actions of the objects at design time or run time program which is then compiled as you find code compiled in C++ or dBASE, or even Pascal. Nor does the container have the kind of logical constructs one finds in standard interpreted systems like Basic. It is useful to keep in mind that Visual Basic, at the time of the present invention's invention, created source code that had to be compiled to be run.

Examiner's paragraph 7, page 10:

Per claims 26, 27, and 53, Smith discloses at least:

means for wrapping standardized objects with additional properties and events beyond those properties and events provided in the standardized object (see at least column 3, lines 32-37; column 4, lines 8-19 and related discussion elsewhere in the specification);

Col. 3, lines 32-37 is a description of the user interface that surrounds the line editor, it is not a description of the kind of encapsulation of a programmable software object referred to in the present application. The user interface described in Smith's invention is technically similar to any dialogue box found in any GUI based program. In the typical dialogue box, there is an edit field and several buttons. Smith does not claim to have invented this kind of user interface. The internal details of such a dialogue box are such that some elements of the interface may be implemented using readily available software objects like buttons. Using such a software object as part of a dialogue box is common practice known to those schooled in art of software development, and is not similar to the way the present invention wraps individual objects with properties and events. The dialogue box puts buttons around an edit box. By contrast, the present invention adds new properties and events to objects in such a way that those newly added properties and events work within the programming system just as all the native properties and events operate. The added properties and events extend the capabilities of the object so the object becomes more manageable within the present invention's environment. This is very different from putting a button next to an edit box, as Smith shows in his invention.

In the present invention individual objects are wrapped with new properties by the present invention's container. This method can be applied both to objects written to a defined specification or custom objects which are found after their creation to need other attributes. The wrapped properties act as part of the object, so if an object comes into the present invention with three standard properties, the present invention might add three more. The object then acts as

if it has six properties, and the present invention properties and the original properties interact with the container in exactly the same manner. This is a form of encapsulation, or sub-classing, of objects, including those written to a defined specification, that was not taught by the prior art.

means for utilizing the additional properties and events to link and sequence the objects (see at least column 4, line 21-61 and related discussion elsewhere in the specification); *and*

Col. 4, lines 21-61 describes the rules which are used to maintain the correct syntax within the line editor. These rules are based on the programming system to which the "Expression Builder" is attached, dBase being specified for the preferred embodiment. The underlying systems did not add or use additional properties the way the present invention does, so the "Expression Builder" would not have the basis for rules to allow them to use such properties either. The only sequencing recited at Col. 4, lines 21-61 is the sequencing of syntactical elements within a single expression according to a set of exiting rules. This is very different from the sequencing of whole objects and managing the relationships between whole objects as taught in the present invention and set forth in the claims.

means for specifying the temporal relationship among standard objects by placing the objects on one or more time lines wherein execution of the object occurs at least partially concurrently and during which property values may be exchanged among the objects and events may be initiated (see at least Figure 2A, "Reports", and related discussion in the specification).

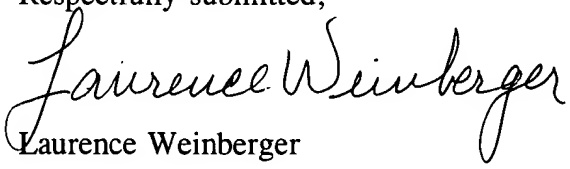
Figure 2A has no reference to Reports, rather Results. There is nothing in the "Expression Builder" line editing system that addressed the instantiation of objects on the basis of time. While it might be possible to create an application with the "Expression Builder" as part of a C++ development system there is on indication that the "Expression Builder" itself is designed for this purpose. Since the Smith invention does not deal with employing objects in an

object oriented programming environment, it can not specify temporal relationships among objects.

Finally, Applicants wish to clearly state that the amendments to the claims (other than that made to the objection noted by the Examiner and to the 35 U.S.C. §112 rejections) presented herein are not in response to the 35 U.S.C. §102(e) rejections made by the Examiner. As Applicants have tried to make clear above, it is their position that the Smith reference does not in any way anticipate their invention, and, therefore, no narrowing or alteration of the scope of the claims in response to the Examiner's rejections is intended. The amendments to the claims are intended by Applicants to more clearly point out and distinctly claim the subject matter of their invention.

Applicants submit that they have addressed all of the Examiner's bases for objection and rejection, and respectfully request that the Examiner withdraw the objections and rejections and permit the patent to issue.

Respectfully submitted,



Laurence Weinberger

Attorney for Applicants
USPTO Reg. No. 27,965
Suite 103
882 S. Matlack St.
West Chester, PA 19382
610-431-1703
610-431-4181 (fax)
larry@lawpatent.com

MARKED UP COPY OF CLAIMS WITH ADDITIONS UNDERLINED AND DELETIONS IN BRACKETS.

1. A computer implemented system employing objects for generating an application script, in which both the objects and the script may be maintained separately, [without the necessity of specifying programatic steps in a character based representation] comprising:

a. means for wrapping [standardized] objects with additional properties and events beyond those properties and events internal to the [provided in the standardized] object; and

b. means for utilizing the additional properties and events to link and sequence the objects [into the application].

2. A computer implemented system employing objects for generating an application script , in which both the objects and the script may be maintained separately, representing [an application] a program structure [without the necessity of specifying programatic programmatic steps in a character based representation] comprising:

a. means for simultaneously displaying a plurality of different representations of the program structure; and

b. means for manipulating the program structure within each of the [four] different representations[;]

wherein the representations of the program structure may be synchronized [among the displays at the election of the user].

3. The system of claim 2 further comprising a means for highlighting the icon for each object [depiction of the objects] in the representations as [those] objects are being instantiated [realized] during application development playback preview.

4. A computer implemented system employing objects and utilizing a script, in which both the objects and the script may be maintained separately, [which does not require the necessity of specifying programatic steps in a character based representation] comprising:

a. a development environment and an interpreting run time [a playback] environment [that have no logical operators]; and

b. means for utilizing objects [,] by specifying property values according to the script[, standard objects].

5. The system of claim 4 further comprising a means for communicating among [standard] objects through the exchange of property values.

6. The system of claim 5 further comprising a means for communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

7. The system of claim 4 further comprising a means for communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

8. A computer implemented system employing objects and utilizing a script, in which both the objects and the script may be maintained separately, [which does not require the necessity of specifying programatic steps in a character based representation] comprising:

a. a development environment and an interpreting run time [a playback] environment that have no logical or arithmetic operators; and

b. means for utilizing objects [,] by specifying property values according to the script[, standard objects].

9. The system of claim 8 further comprising a means for communicating among [standard] objects through the exchange of property values.

10. The system of claim 9 further comprising a means for communicating among [standard]

objects wherein an event generated by an object triggers an instance of [an] another object.

11. The system of claim 8 further comprising a means for communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

12. A computer implemented system employing objects and utilizing a script, in which both the objects and the script may be maintained separately, [which does not require the necessity of specifying programatic steps in a character based representation] comprising:

a. a development environment and an interpreting run time [a playback] environment that have no definable data structure architecture; and

b. means for utilizing objects [,] by specifying property values according to the script [, standard] objects].

13. The system of claim 12 further comprising a means for communicating among [standard] objects through the exchange of property values.

14. The system of claim 13 further comprising a means for communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

15. The system of claim 12 further comprising a means for communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

16. The system of claim 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, or 15 further comprising a means for adding additional programming constructs by employing [standard] objects that perform the function [work] of programming constructs wherein unlimited expansion of program capabilities is achieved.

17. A computer implemented system employing objects and interpreting a script in which both the objects and the script may be maintained separately, comprising:

a. a run time program [that has no logical operators]; and

b. means for utilizing [standard] objects according to the script [by identifying the objects and specifying property values].

18. A computer implemented system employing objects and interpreting a script in which both the objects and the script may be maintained separately, comprising:

a. a run time program that has [no] neither arithmetic nor logical operators; and

b. means for utilizing [standard] objects according to the script [by identifying the objects and specifying property values].

19. A computer implemented system employing objects and interpreting a script in which both the objects and the script may be maintained separately,, comprising:

a. a run time program that has no definable data structure architecture; and

b. means for utilizing [standard] objects according to the script [by identifying the objects and specifying property values].

20. A computer implemented development and run time system employing objects which utilizes a script, in which both the objects and the script may be maintained separately, utilizing a minimum set of core functionalities comprising:

a. means for instantiating objects;

b. means for integrating objects;

c. means for sequencing objects; and

d. means for providing communication among objects[; and

e. means for synchronizing views]

wherein the [displayed] functionalities performed by the system during execution are determined by the [choice of] objects used and the script. [manner of their implementation in the system.]

21. A computer implemented run time system employing objects utilizing a minimum set of core functionalities which interprets a script, in which both the objects and the script may be maintained separately, comprising:

a. means for instantiating objects;

b. means for integrating objects;

c. means for sequencing objects; and

d. means for providing communication among objects;

wherein the [displayed] functionalities performed by the system during execution are determined by the [choice of] objects used and the script. [manner of their implementation in the system.]

22. A computer implemented system for [arranging] employing objects , having property values and event connections, which can be set in time and turned on or off of a visually perceptible display device comprising:

a. means for setting the values of properties and connecting events;

b. means for recording and maintaining a history of a plurality of properties settings and event connections as the settings and connections are changed; and

c. means for traversing the history one change at a time

wherein the property values and event connections may be edited from any point in the history.

23. A computer implemented run time system employing objects which [does not require the necessity of specifying programatic steps in a character based representation that] interprets a script containing property values and event settings, in which both the objects and the script may be maintained separately, and dynamically executes objects comprising:

a. means for wrapping [standardized] objects with additional properties and events beyond those properties and events internal to the [provided in the standardized] objects;

b. means for utilizing the additional properties and events to link and sequence the objects; and

c. means for reading one or more sets of property values and event settings maintained separately from the run time system and the objects

wherein the execution of the objects is determined [governed] by the property values and event settings in the script [property values].

24. The system of claim 23 further comprising means for adding programming constructs [and] or sub-languages utilizing objects.

25. A computer implemented system which interprets a script containing property values and event settings, which may be maintained separately, [which does not require the necessity of specifying programatic steps in a character based representation] that distributes processing to objects, provides and manages data flow among objects, and manages the execution [scheduling] of objects comprising:

a. means for wrapping [standardized] objects with additional properties and events beyond those properties and events internal to the [provided in the standardized] object; and

b. means for utilizing the additional properties and events to link and sequence the objects [;

c. means for specifying property values; and

d. means for saving the property values to a separate file]

wherein the run time execution of the objects is determined by [the] property values and events.

26. A computer implemented system employing objects which implements parallel processing [without the necessity of specifying programatic steps in a character based representation] comprising:

a. means for wrapping [standardized] objects with additional properties and events beyond those properties and events provided internal to the [in the standardized] object;

b. means for utilizing the additional properties and events to link and sequence the objects; and

c. means for specifying the temporal relationship among [standard] objects by placing the objects on one or more time lines

wherein execution of the objects occurs at least partially concurrently and during which property values may be exchanged among the objects and events may be initiated.

27. An object oriented programming [A] computer implemented system in which the function of programming constructs is achieved by dynamically executing [utilization of standard] objects comprising:

a. means for wrapping [standardized] objects with additional properties and events beyond those properties and events provided internal to the [in the standardized] object;

b. means for utilizing the additional properties and events to link and sequence the objects [into the application]; and

c. means for specifying a list of property values and event settings

wherein the execution of the objects is determined by [a] the list of property values and event settings.

28. A computer implemented software method employing objects for generating an application script, in which both the objects and the script may be maintained separately,

[without the necessity of specifying programatic steps in a character based representation] comprising the steps of:

a. wrapping [standardized] objects with additional properties and events beyond those properties and events internal to the [provided in the standardized] object; and

b. utilizing the additional properties and events to link and sequence the objects [into the application].

29. A computer implemented software method employing objects for generating an application script, in which both the objects and the script may be maintained separately, representing [an application] a program structure [without the necessity of specifying programatic programmatic steps in a character based representation] comprising the steps of:

a. simultaneously displaying a plurality of different representations of the program structure; and

b. means for manipulating the program structure within each of the [four] different representations[;]

wherein the representations of the program structure may be synchronized [among displays at the election of the user].

30. The software method of claim 29 further comprising the step of highlighting the icon for each object [depiction of the objects] in the representations as [those] objects are being instantiated [realized] during application development run time [playback] preview.

31. A computer implemented software method employing objects and utilizing a script, in which both the objects and the script may be maintained separately, [for programming a computer which does not require the necessity of specifying programatic steps in a character based representation] comprising the steps of:

a. utilizing a development environment and an interpreting run time [a playback] environment [that have no logical operators]; and

b. utilizing objects [,] by specifying property values according to the script[, standard objects].

32. The software method of claim 31 further comprising the step of communicating among [standard] objects through the exchange of property values.

33. The software method of claim 32 further comprising the step of communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

34. The software method of claim 31 further comprising the step of communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

35. A computer implemented software method employing objects and utilizing a script, in which both the objects and the script may be maintained separately, [for programmming a computer which does not require the necessity of specifying programatic steps in a character based representation] comprising the steps of:

a. utilizing a development environment and an interpreting run time [a playback] environment that have no logical or arithmetic operators; and

b. utilizing objects [,] by specifying property values according to the script[, standard objects].

36. The software method of claim 35 further comprising the step of communicating among [standard] objects through the exchange of property values.

37. The software method of claim 36 further comprising the step of communicating among

[standard] objects wherein an event generated by an object triggers an instance of [an] another object.

38. The software method of claim 35 further comprising the step of communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

39. A computer implemented software method employing objects and utilizing a script, in which both the objects and the script may be maintained separately, [which does not require the necessity of specifying programatic steps in a character based representation] comprising the steps of:

a. utilizing a development environment and an interpreting run time [a playback] environment that have no definable data structure architecture; and

b. utilizing objects [,] by specifying property values according to the script [, standard] objects].

40. The software method of claim 39 further comprising the step of communicating among [standard] objects through the exchange of property values.

41. The software method of claim 40 further comprising the step of communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

42. The software method of claim 39 further comprising the step of communicating among [standard] objects wherein an event generated by an object triggers an instance of [an] another object.

43. The software method of claim 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, or 42 further comprising the step of adding additional programming constructs by employing [standard] objects

that perform the function [work] of programming constructs wherein unlimited expansion of program capabilities is achieved.

44. A computer implemented software method employing objects and interpreting a script in which both the objects and the script may be maintained separately, for executing an application comprising the steps of:

a. utilizing a run time program [that has no logical operators]; and

b. utilizing [standard] objects according to the script [by identifying the objects and specifying property values].

45. A computer implemented software method employing objects and interpreting a script in which both the objects and the script may be maintained separately, for executing an application comprising the steps of:

a. utilizing a run time program that has [no] neither arithmetic nor logical operators; and

b. utilizing [standard] objects according to the script [by identifying the objects and specifying property values].

46. A computer implemented software method employing objects and interpreting a script in which both the objects and the script may be maintained separately, for executing an application comprising the steps of:

a. utilizing a run time program that has no definable data structure architecture; and

b. utilizing [standard] objects according to the script [by identifying the objects and specifying property values].

47. A computer implemented development and run time software method employing objects for developing and executing an application which utilizes a script, in which both the objects and

the script may be maintained separately, and utilizing a minimum set of core functionalities comprising the steps of:

a. instantiating objects;

b. integrating objects;

c. sequencing objects; and

d. providing communication among objects[; and

e. synchronizing views]

wherein the [displayed] functionalities performed by the software method during execution are determined by the [choice of] objects used and the script [manner of their implementation in the system].

48. A computer implemented run time software method employing objects for executing an application utilizing a minimum set of core functionalities which interprets a script, in which both the objects and the script may be maintained separately, comprising the steps of:

a. instantiating objects;

b. integrating objects;

c. sequencing objects; and

d. providing communication among objects;

wherein the [displayed] functionalities performed by the software method during execution are determined by the [choice of] objects used and the script. [manner of their implementation in the system.]

49. A computer implemented software method for [arranging] employing objects , having property values and event connections, which can be set in time and turned on or off of a visually perceptible display device comprising the steps of:

a. setting the values of properties and connecting events;

b. recording and maintaining a history of a plurality of properties settings and event connections as the settings and connections are changed; and

c. traversing the history one change at a time

wherein the property values and event connections may be edited from any point in the history.

50. A computer implemented run time software method employing objects which [does not require the necessity of specifying programatic steps in a character based representation that] interprets a script containing property values and event settings, in which both the objects and the script may be maintained separately, and dynamically executes the objects comprising the steps of:

a. wrapping [standardized] objects with additional properties and events beyond those properties and events internal to the [provided in the standardized] objects;

b. utilizing the additional properties and events to link and sequence the objects; and

c. reading one or more sets of property values and event settings maintained separately from the run time system and the objects

wherein the execution of the objects is determined [governed] by the property values and event settings in the script [property values].

51. The software method of claim 50 further comprising the step of adding programming constructs [and] or sub-languages utilizing objects.

52. A computer implemented software method which interprets a script containing property values and event settings, which may be maintained separately, [which does not require the necessity of specifying programatic steps in a character based representation] that distributes

processing to objects, provides and manages data flow among objects, and manages the execution [scheduling] of objects comprising the steps of:

a. wrapping [standardized] objects with additional properties and events beyond those properties and events internal to the [provided in the standardized] object; and

b. utilizing the additional properties and events to link and sequence the objects [;

c. means for specifying property values; and

d. saving the property values to a separate file]

wherein the [run time] execution of the objects is determined by the property values and events.

53. A computer implemented software method employing objects which implements parallel processing [without the necessity of specifying programatic steps in a character based representation] comprising the steps of:

a. wrapping [standardized] objects with additional properties and events beyond those properties and events provided internal to the [in the standardized] object;

b. utilizing the additional properties and events to link and sequence the objects; and

c. specifying the temporal relationship among [standard] objects by placing the objects on one or more time lines

wherein execution of the objects occurs at least partially concurrently and during which property values may be exchanged among the objects and events may be initiated.

54. A computer implemented object oriented software programming method in which the function of programming constructs is achieved by dynamically executing [utilization of standard] objects comprising the steps of:

a. wrapping [standardized] objects with additional properties and events beyond those

properties and events provided internal to the [in the standardized] object;

b. utilizing the additional properties and events to link and sequence the objects [into the application]; and

c. specifying a list of property values and event settings

wherein the execution of the objects is determined by [a] the list of property values and event settings.